1. What modifiers are implicitly applied to all interface methods? (Choose all that apply)

A. protected
B. public
C. static
D. void
E. abstract
F. default

B.
All interface methods are implicitly public, so option B is correct and option A is not.
Interface methods **may be declared as** static or default but are never implicitly added, so options C and F are incorrect.
Option D is incorrect—**void is not a modifier; it is a return type.**
Option E is a tricky one, because prior to Java 8 all interface methods would be assumed to be abstract.
Since Java 8 now includes default and static methods and they are never abstract, you cannot assume the abstract modifier will be implicitly applied to all methods by the compiler.

2. What is the output of the following code?

```
1: class Mammal {
2: public Mammal(int age) {
3: System.out.print("Mammal");
4: }
5: }
6: public class Platypus extends Mammal {
6a: super(int age);
7: public Platypus() {
8: System.out.print("Platypus");
9: }
10: public static void main(String[] args) {
11: new Mammal(5);
12: }
13: }
```

A. Platypus
B. Mammal
C. PlatypusMammal
D. MammalPlatypus
E. The code will not compile because of line 8.
F. The code will not compile because of line 11.

E.
The code will not compile because the parent class Mammal doesn't define a no-argument constructor,

so the first line of a Platypus constructor should be an explicit call to super(int age).
If there was such a call, then the output would be **MammalPlatypus**, since the super constructor is executed before the child constructor.

3. Which of the following statements can be inserted in the blank line so that the code will compile successfully? (Choose all that apply)

```
public interface CanHop {}
public class Frog implements CanHop {
	public static void main(String[] args) {
		_____ frog = new TurtleFrog();
	}
}
public class BrazilianHornedFrog extends Frog {}
public class TurtleFrog extends Frog {}
```

A. Frog
B. TurtleFrog
C. BrazilianHornedFrog
D. CanHop
E. Object
F. Long

A, B, D, E.
The blank can be filled with any class or interface that is a supertype of TurtleFrog.
Option A is a superclass of TurtleFrog, and option B is the same class, so both are correct.
BrazilianHornedFrog is not a superclass of TurtleFrog, so option C is incorrect.
TurtleFrog inherits the CanHope interface, so option D is correct.
All classes inherit Object, so option E is correct.
Finally, **Long is an unrelated class** that is not a superclass of TurtleFrog, and is therefore incorrect

4. Which statement(s) are correct about the following code? (Choose all that apply)

```
public class Rodent {
	protected static Integer chew() throws Exception {
		System.out.println("Rodent is chewing");
		return 1;
	}
}
public class Beaver extends Rodent {
	public Number chew() throws RuntimeException {
		System.out.println("Beaver is chewing on wood");
		return 2;
	}
}
```

A. It will compile without issue.
B. It fails to compile because the type of the exception the method throws is a subclass of the type of exception the parent method throws.
C. It fails to compile because the return types are not covariant.
D. It fails to compile because the method is protected in the parent class and public in the subclass.
E. It fails to compile because of a static modifier mismatch between the two methods.

C, E.
The code doesn't compile, so **option A is incorrect**.
Option B is also not correct because the rules for overriding a method allow a subclass to define a method with an exception that is a subclass of the exception in the parent method.
Option C is correct because the return types are **not covariant**; in particular, Number is not a subclass of Integer.
Option D is incorrect because the subclass defines a method that is more accessible than the method in the parent class, which is allowed.
Finally, option E is correct because the method is **declared as static** in the parent class and not so in the child class.
For nonprivate methods in the parent class, both methods must use static (hide) or neither should use static (override)

5. Which of the following may only be hidden and not overridden? (Choose all that apply)

A. private instance methods
B. protected instance methods
C. public instance methods
D. static methods
E. public variables
F. private variables

A, D, E, F.
First off, options B and C are incorrect because protected and public methods may be overridden, not hidden.
Option A is correct because private methods are always hidden in a subclass.
Option D is also correct because static methods cannot be overridden, only hidden.
Options E and F are correct because variables may only be hidden, regardless of the access modifier

6. Choose the correct statement about the following code:

```
1: interface HasExoskeleton {
        2: abstract int getNumberOfSections();
3: }
4: abstract class Insect implements HasExoskeleton {
        5: abstract int getNumberOfLegs();
6: }
```

```
7: public class Beetle extends Insect {
        8: int getNumberOfLegs() { return 6; }
9: }
```

A. It compiles and runs without issue.
B. The code will not compile because of line 2.
C. The code will not compile because of line 4.
D. The code will not compile because of line 7.
E. It compiles but throws an exception at runtime.

D.
The code fails to compile because Beetle, the first concrete subclass, doesn't implement getNumberOfSections(),
which is inherited as an abstract method; therefore, option D is correct.
Option B is incorrect because there is nothing wrong with this interface method definition.
Option C is incorrect because an abstract class is not required to implement any abstract methods, including those inherited from an interface.
Option E is incorrect because the code fails at compilation-time

7. Which of the following statements about polymorphism are true? (Choose all that apply)

A. A reference to an object may be cast to a subclass of the object without an explicit cast.
B. If a method takes a superclass of three objects, then any of those classes may be passed as a parameter to the method.
C. A method that takes a parameter with type java.lang.Object will take any reference.
D. All cast exceptions can be detected at compile-time.
E. By defining a public instance method in the superclass, you guarantee that the specific method will be called in the parent class at runtime.

B, C.
A reference to an object requires an explicit cast if referenced with a subclass, so option A is incorrect.
If the cast is to a superclass reference, then an explicit cast is not required.
Because of polymorphic parameters, if a method takes the superclass of an object as a parameter,
then any subclass references may be used without a cast, so option B is correct.
All objects extend java.lang.Object, so if a method takes that type, any valid object, including null, may be passed; therefore, option C is correct.
Some cast exceptions can be detected as errors at compile-time, but others can only be detected at runtime, so D is incorrect.
Due to the nature of polymorphism, a public instance method can be overridden in a subclass and calls to it will be replaced even in the superclass it was defined, so E is incorrect

8. Choose the correct statement about the following code:

```
1: public interface Herbivore {
        2: int amount = 10;
        3: public static void eatGrass();
        4: public int chew() {
                5: return 13;
        6: }
7: }
```

A. It compiles and runs without issue.
B. The code will not compile because of line 2.
C. The code will not compile because of line 3.
D. The code will not compile because of line 4.
E. The code will not compile because of lines 2 and 3.
F. The code will not compile because of lines 3 and 4.

F.
The interface variable amount is correctly declared, with **public and static** being assumed and automatically inserted by the compiler, so option B is incorrect.
The method declaration for eatGrass() on line 3 is incorrect because the method has been marked as static but no method body has been provided.
The method declaration for chew() on line 4 is also incorrect, since an interface method that provides a body must be marked as **default or static** explicitly.
Therefore, option F is the correct answer since this code contains two compile-time errors.

9. Choose the correct statement about the following code:

```
1: public interface CanFly {
2: void fly();
3: }
4: interface HasWings {
5: public abstract Object getWindSpan();
6: }
7: abstract class Falcon implements CanFly, HasWings {
8: }
```

A. It compiles without issue.
B. The code will not compile because of line 2.
C. The code will not compile because of line 4.
D. The code will not compile because of line 5.
E. The code will not compile because of lines 2 and 5.
F. The code will not compile because the class Falcon doesn't implement the interface methods.

A.
Although the definition of methods on lines 2 and 5 vary, both will be converted to public abstract by the compiler.

Line 4 is fine, because an interface can have public or default access.
Finally, the class Falcon doesn't need to implement the interface methods because it is marked as abstract.
Therefore, the code will compile without issue

10. Which statements are true for both abstract classes and interfaces? (Choose all that apply)

A. All methods within them are assumed to be abstract.
B. Both can contain public static final variables.
C. Both can be extended using the extend keyword.
D. Both can contain default methods.
E. Both can contain static methods.
F. Neither can be instantiated directly.
G. Both inherit java.lang.Object.

B, C, E, F.
Option A is wrong, because an abstract class may contain concrete methods.
Since Java 8, interfaces may also contain concrete methods in form of static or default methods.
Although all variables in interfaces are assumed to be public static final, abstract classes may contain them as well, so option B is correct.
Both abstract classes and interfaces can be extended with the extends keyword, so option C is correct.
Only interfaces can contain default methods, so option D is incorrect.
Both abstract classes and interfaces can contain static methods, so option E is correct.
Both structures require a concrete subclass to be instantiated, so option F is correct.
Finally, though an instance of an object that implements an interface inherits java.lang. Object, the interface itself doesn't; otherwise, Java would support multiple inheritance for objects, which it doesn't.
Therefore, option G is incorrect

11. What modifiers are assumed for all interface variables? (Choose all that apply)

D. public
A. protected
F. private
B. static
E. final
C. abstract

A, D, E.
Interface variables are assumed to be public static final; therefore, options A, D, and E are correct.
Options B and C are incorrect because interface variables must be public—interfaces are implemented by classes, not inherited by interfaces.

Option F is incorrect because variables can never be abstract

12. What is the output of the following code?

```
1: interface Nocturnal {
        2: default boolean isBlind() { return true; }
3: }
4: public class Owl implements Nocturnal {
        5: public boolean isBlind() { return false; }
        6: public static void main(String[] args) {
                7: Nocturnal nocturnal = (Nocturnal)new Owl();
                8: System.out.println(nocturnal.isBlind());
        9: }
10: }
```

A. true
B. false
C. The code will not compile because of line 2.
D. The code will not compile because of line 5.
E. The code will not compile because of line 7.
F. The code will not compile because of line 8.

B.
This code compiles and runs without issue, outputting false, so option B is the correct answer.
The first declaration of isBlind() is as a default interface method, assumed public.
The second declaration of isBlind() correctly overrides the default interface method.
Finally, the newly created Owl instance may be automatically cast to a Nocturnal reference without an explicit cast, although adding it doesn't break the code

13. What is the output of the following code?

```
1: class Arthropod
        2: public void printName(double input) { System.out.print("Arthropod"); }
3: }
4: public class Spider extends Arthropod {
        5: public void printName(int input) { System.out.print("Spider"); }
        6: public static void main(String[] args) {
                7: Spider spider = new Spider();
                8: spider.printName(4);
                9: spider.printName(9.0);
        10: }
11: }
```

A. SpiderArthropod
B. ArthropodSpider
C. SpiderSpider
D. ArthropodArthropod
E. The code will not compile because of line 5.
F. The code will not compile because of line 9.

A.
The code compiles and runs without issue, so options E and F are incorrect.
The printName() method is an overload in Spider, not an override, so both methods may be called.
The call on line 8 references the version that takes an int as input defined in the Spider class, and the call on line 9 references the version in the Arthropod class that takes a double.
Therefore, **SpiderArthropod** is output and option A is the correct answer

14. Which statements are true about the following code? (Choose all that apply)

```
1: interface HasVocalCords {
        2: public abstract void makeSound();
3: }
4: public interface CanBark extends HasVocalCords {
        5: public void bark();
6: }
```

A. The CanBark interface doesn't compile.
B. A class that implements HasVocalCords must override the makeSound() method.
C. A class that implements CanBark inherits both the makeSound() and bark() methods.
D. A class that implements CanBark only inherits the bark() method.
E. An interface cannot extend another interface.

C.
The code compiles without issue, so option A is wrong.
Option B is incorrect, since an abstract class could implement HasVocalCords without the need to override the makeSound() method.
Option C is correct;
any class that implements CanBark automatically inherits its methods, as well as any inherited methods defined in the parent interface.
Because option C is correct, it follows that option D is incorrect.
Finally, **an interface can extend multiple interfaces**, so option E is incorrect

15. Which of the following is true about a concrete subclass? (Choose all that apply)

A. A concrete subclass can be declared as abstract.
B. A concrete subclass must implement **all inherited abstract methods**.
C. A concrete subclass must implement all methods defined in an inherited interface.
D. A concrete subclass cannot be marked as final.

E. Abstract methods cannot be overridden by a concrete subclass.

B.
Concrete classes are, by definition, not abstract, so option A is incorrect.
A concrete class must implement all inherited abstract methods, so option B is correct.
Option C is incorrect; a superclass may have already implemented an inherited interface, so the concrete subclass would not need to implement the method.
Concrete classes can be both final and not final, so option D is incorrect.
Finally, abstract methods **must be** overridden by a concrete subclass, so option E is incorrect

16. What is the output of the following code?

```
1: abstract class Reptile {
        2: public final void layEggs() { System.out.println("Reptile laying eggs");
        }
        3: public static void main(String[] args) {
                4: Reptile reptile = new Lizard();
                5: reptile.layEggs();
        6: }
7: }
8: public class Lizard extends Reptile {
        9: public void layEggs() { System.out.println("Lizard laying eggs"); }
10: }
```

A. Reptile laying eggs
B. Lizard laying eggs
C. The code will not compile because of line 4.
D. The code will not compile because of line 5.
E. The code will not compile because of line 9.

E.
The code doesn't compile, so options A and B are incorrect.
The issue with line 9 is that layEggs() is marked as final in the superclass Reptile, which means it cannot be overridden.
There are no errors on any other lines, so options C and D are incorrect

17. What is the output of the following code?

```
1: public abstract class Whale {
        2: public abstract void dive() {};
        3: public static void main(String[] args) {
                4: Whale whale = new Orca();
                5: whale.dive();
                6: }
7: }
8: class Orca extends Whale {
```

```
        9: public void dive(int depth) { System.out.println("Orca diving"); }
10: }
```
A. Orca diving
B. The code will not compile because of line 2.
C. The code will not compile because of line 8.
D. The code will not compile because of line 9.
E. The output cannot be determined from the code provided.

B.
This may look like a complex question, but it is actually quite easy.
Line 2 contains an invalid definition of an abstract method.
Abstract methods **cannot contain a body**, so the code will not compile and option B is the correct answer.
If the body {} was removed from line 2, the code would still not compile,
although it would be line 8 that would throw the compilation error.
Since dive() in Whale is abstract and Orca extends Whale,
then it must implement an overridden version of dive().
The method on line 9 is an overloaded version of dive(),
not an overridden version, so Orca is an invalid subclass and will not compile

18. What is the output of the following code? (Choose all that apply)

```
1: interface Aquatic {
        2: public default int getNumberOfGills(int input) { return 2; }
3: }
4: public class ClownFish implements Aquatic {
        5: public String getNumberOfGills() { return "4"; }
        6: public String getNumberOfGills(int input) { return "6"; }
        7: public static void main(String[] args) {
                8: System.out.println(new ClownFish().getNumberOfGills(-1));
        9: }
10: }
```
A. 2
B. 4
C. 6
D. The code will not compile because of line 5.
E. The code will not compile because of line 6.
F. The code will not compile because of line 8.

E.
The code doesn't compile because line 6 contains an incompatible override of the getNumberOfGills(int input) method defined in the Aquatic interface.
In particular, int and String are not covariant returns types, since int is not a subclass of String.
Note that line 5 compiles without issue; getNumberOfGills() is an overloaded method that is not related to the parent interface method that takes an int value

19. Which of the following statements can be inserted in the blank so that the code will compile successfully? (Choose all that apply)

```
public class Snake {}
public class Cobra extends Snake {}
public class GardenSnake {}
public class SnakeHandler {
        private Snake snake;
        public void setSnake(Snake snake) { this.snake = snake; }
        public static void main(String[] args) {
                new SnakeHandler().setSnake( _____);
        }
}
```

A. new Cobra()
B. new GardenSnake()
C. new Snake()
D. new Object()
E. new String("Snake")
F. null

A, C, F.
First off, Cobra is a subclass of Snake, so option A can be used.
GardenSnake is not defined as a subclass of Snake, so it cannot be used and option B is incorrect.
The class Snake is not marked as abstract, so it can be instantiated and passed, so option C is correct.
Next, Object is a superclass of Snake, not a subclass, so it also cannot be used and option D is incorrect.
The class String is unrelated in this example, so option E is incorrect.
Finally, a null value can always be passed as an object value, regardless of type, so option F is correct

20. What is the result of the following code?

```
1: public abstract class Bird {
        2: private void fly() { System.out.println("Bird is flying"); }
        3: public static void main(String[] args) {
                4: Bird bird = new Pelican();
                5: bird.fly();
        6: }
7: }
8: class Pelican extends Bird {
        9: protected void fly() { System.out.println("Pelican is flying"); }
10: }
```

A. Bird is flying
B. Pelican is flying
C. The code will not compile because of line 4.
D. The code will not compile because of line 5.
E. The code will not compile because of line 9.

A.
The code compiles and runs without issue, so options C, D, and E are incorrect.
The trick here is that the method fly() is marked as private in the parent class Bird, which means it may only be hidden, not overridden.
With hidden methods, the specific method used depends on where it is referenced. Since it is referenced within the Bird class, the method declared on line 2 was used, and option A is correct.
Alternatively, if the method was referenced within the Pelican class, or if the method in the parent class was marked as protected and overridden in the subclass, then the method on line 9 would have been used.